**Санкт-Петербургский государственный УНИВЕРСИТЕТ**

**Факультет прикладной математики и процессов управления**

**отчет**

**по лабораторной работе №6**

**по дисциплине «Алгоритмы и структуры данных»**

**на тему «Приближенное вычисление площади фигуры методом Монте-Карло»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 22Б16 |  | Шувалов Ф.В. |
| Преподаватель |  | Дик А.Г. |

**Санкт-Петербург**

**2023 г.**

**Оглавление**

[1. Цель работы 3](#_Toc163902451)

[3. Введение 3](#_Toc163902452)

[4. Схема выполнения алгоритма 3](#_Toc163902453)

[6. Представление программы на языке программирования 5](#_Toc163902454)

[7. Раскрытие смысла работы алгоритма 5](#_Toc163902455)

[8. Контрольный пример 5](#_Toc163902456)

[9. Вывод 7](#_Toc163902457)

[10. Литература 7](#_Toc163902458)

# **Цель работы**

Цель работы – изучение метода Монте-Карло (метода статистических испытаний) на примере вычисления площади фигуры.

1. **Задачи**
   * 1. Составить и отладить программу определения площади фигуры методом Монте-Карло в соответствии с индивидуальным заданием.
     2. Вычислить методом Монте-Карло определенный интеграл. Сравните результат со значением, полученным аналитическим путем при значениях N=10, 100, 1000, 10000, 100000, 1000000. Выразите относительную погрешность метода Монте-Карло при каждом значении N

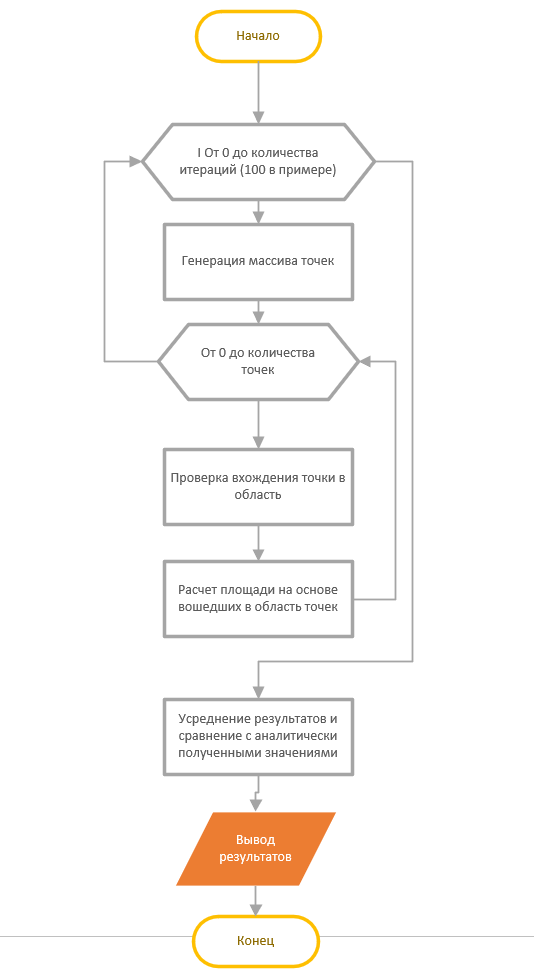
# **Теоретическая часть**

Применим метод статических испытаний или метод Монте-Карло к задаче вычисления площади геометрической фигуры на плоскости. Метод заключается в следующем. Поместим данную фигуру в квадрат и будем наугад бросать точки в этот квадрат. Будем исходить из того, что чем больше площадь фигуры, тем чаще в нее будут попадать точки. Таким образом, при большом числе N точек, наугад выбранных внутри квадрата, доля точек, содержащихся в данной фигуре k, приближенно равна отношению площади этой фигуры и площади квадрата: Если площадь квадрата равна S0 (4х4=16 отрезки [-2;2]) и в результате N испытаний, из которых при k исходах случайные точки оказались внутри фигуры, то площадь фигуры S будет определяться выражением:

Относительная погрешность метода Монте-Карло определяться выражением:

# **Схема выполнения алгоритма**

На рисунке 4.1 представлена блок-схема алгоритма.

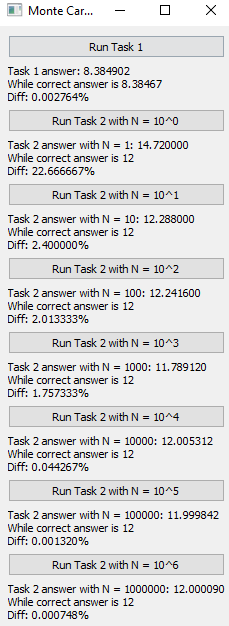


*Рисунок 4.1 Блок-схема основного алгоритма*

# **Представление программы на языке программирования**

Алгоритм обхода дерева реализован на языке Golang в виде функции task.

# **Контрольный пример**



*Рисунок 6.1 Визуализация программы*

На рисунке 10.1 изображен пользовательский интерфейс программы и результат работы алгоритма.

# **Анализ результатов работы алгоритма**

По результатам, полученным на основе тестирования программы (см. “Контрольный пример”), видно, что метод Монте-Карло становится более точным с увеличением количества точек. Все представленные результаты являются средними значениями, полученными из 100 испытаний, что помогает минимизировать случайные отклонения и дает более стабильную оценку точности метода. Эта зависимость между количеством точек и точностью результата можно охарактеризовать как линейную, то есть, с увеличением числа точек пропорционально увеличивается точность вычислений. Такой подход демонстрирует надежность и эффективность метода Монте-Карло при решении задач, требующих статистического анализа и моделирования.

# **Вывод**

При выполнении данной работы были получены все необходимые навыки и знания по реализации метода Монте-Карло. В ходе реализации задачи была создана программа для визуализации его работы.

# **Литература**

* + 1. Документация библиотеки PyQT [Электронный ресурс] - URL: <https://doc.qt.io/qtforpython-6/> (дата обращения: 25.09.2023)

1. **Листинг**
2. package main  
     
   import (  
    "fmt"  
    "math"  
    "math/rand"  
    "time"  
     
    "github.com/therecipe/qt/widgets"  
   )  
     
   type point struct {  
    x, y float64  
   }  
     
   func RandomNumberGenerator() \*rand.Rand {  
    s1 := rand.NewSource(time.Now().UnixNano())  
    r1 := rand.New(s1)  
    return r1  
   }  
     
   func RandArray(from, to float64, N int) (a []point) {  
    rng := RandomNumberGenerator()  
    a = make([]point, N)  
    for i := 0; i < N; i++ {  
    a[i] = point{rng.Float64()\*(to-from) + from, rng.Float64()\*(to-from) + from}  
    }  
    return  
   }  
     
   func task(pStart, pEnd float64, N, times int, BoolFunc func(dot point) bool, CorrectAns float64) (ans, diff float64) {  
    ans = 0.0  
    for i := 0; i < times; i++ {  
    dots := RandArray(pStart, pEnd, N)  
    correct := 0  
    for j := 0; j < len(dots); j++ {  
    if BoolFunc(dots[j]) {  
    correct += 1  
    }  
    }  
    ans += math.Abs(float64(correct) / float64(N) \* math.Pow(pStart-pEnd, 2))  
    }  
    return ans / float64(times), math.Abs((ans/float64(times) - CorrectAns) / CorrectAns \* 100)  
   }  
     
   func dotIsInFirstArea(dot point) bool {  
    return (-1\*math.Pow(dot.x, 3)-5\*math.Pow(dot.y, 3) < 2) && (-dot.x+dot.y < 2) && (math.Abs(dot.x) < 2) && (math.Abs(dot.y) < 2)  
   }  
     
   func dotIsInSecondArea(dot point) bool {  
    return (dot.y < math.Pow(dot.x, 1/3)) && (math.Abs(dot.x) < 8) && (math.Abs(dot.y) < 8)  
   }  
     
   func main() {  
    // Создание приложения Qt  
    app := widgets.NewQApplication(len([]string{}), []string{})  
     
    // Создание основного окна  
    window := widgets.NewQMainWindow(nil, 0)  
    window.SetWindowTitle("Monte Carlo Simulation")  
     
    // Создание центрального виджета  
    centralWidget := widgets.NewQWidget(nil, 0)  
    window.SetCentralWidget(centralWidget)  
     
    // Создание компоновки  
    layout := widgets.NewQVBoxLayout()  
     
    // Создание кнопки для выполнения задачи 1  
    task1Button := widgets.NewQPushButton2("Run Task 1", nil)  
    task1Result := widgets.NewQLabel2("", nil, 0)  
     
    // Обработка нажатия кнопки  
    task1Button.ConnectClicked(func(checked bool) {  
    ans, diff := task(-2, 2, int(math.Pow10(6)), 100, dotIsInFirstArea, 8.38467)  
    resultText := fmt.Sprintf("Task 1 answer: %f\nWhile correct answer is 8.38467\nDiff: %f%%", ans, diff)  
    task1Result.SetText(resultText)  
    })  
     
    // Добавление кнопки и метки в компоновку  
    layout.AddWidget(task1Button, 0, 0)  
    layout.AddWidget(task1Result, 0, 0)  
     
    // Создание кнопок и меток для задач 2 с различными N  
    for i := 0; i < 7; i++ {  
    task2Button := widgets.NewQPushButton2(fmt.Sprintf("Run Task 2 with N = 10^%d", i), nil)  
    task2Result := widgets.NewQLabel2("", nil, 0)  
    N := int(math.Pow10(i))  
     
    task2Button.ConnectClicked(func(checked bool) {  
    ans, diff := task(-2, 2, N, 100, dotIsInSecondArea, 12)  
    resultText := fmt.Sprintf("Task 2 answer with N = %d: %f\nWhile correct answer is 12\nDiff: %f%%", N, ans, diff)  
    task2Result.SetText(resultText)  
    })  
     
    layout.AddWidget(task2Button, 0, 0)  
    layout.AddWidget(task2Result, 0, 0)  
    }  
     
    // Установка компоновки центрального виджета  
    centralWidget.SetLayout(layout)  
     
    // Отображение окна  
    window.Show()  
     
    // Запуск цикла обработки событий  
    app.Exec()  
   }